Week 8: Metrics in SE for SPM

Rachelle Bosua

rachelle.bosua@unimelb.edu.au

www.lms.unimelb.edu.au

Outline

- Individual Assignment – feedback
- Metrics in SE and SPM

Individual Ass. feedback

1) What we wanted
2) Plagiarism and collusion
3) General feedback on student answers

What we wanted...

- Your were given *an argument* based on an SRS and a SPMP
  - Wanted a critique and reflection
  - Asked three questions:
    - Risks
    - Characteristics, features and requirements making if difficult
    - Thoughts on the SDLC proposed

Plagiarism – 1)

- Plagiarism is:

"*the practice of taking someone else's work or ideas and passing them off as one's own*"

- synonyms: copying, piracy, theft, stealing, appropriation
- acknowledge others through citation
- use *your own words*

Plagiarism – 2)

- Serious misconduct in academia
- Turnitin – index
- Colour coding – high index
- Please refrain from 'poaching' words
- Links:
  UNIMELB:
  https://airport.unimelb.edu.au/gate1/writing/plagiarism/
- http://www.smh.com.au/national/education/cheating-at-major-australian-universities-may-be-easier-than-many-realise-20160122-gmbq30.html

General feedback

- Good structure – essential for any writing
- Risks – what *negatively impacts* on the project
- Characteristics, attributes or 'things' adding to difficulty level – what makes this challenging
- Would any alternative SDLC work?

METRICS

Monitoring and SPM

- Be able to measure and analyse info.

1) Estimates for future performance
2) Compare actual against predictions
3) Assess quality of outputs & activities

Measurement – 1)

- Measurement fundamental to any Eng discipline

*"When you can measure what you are speaking about and express it in numbers, you **know something about it**; **but** when you cannot measure, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science"*

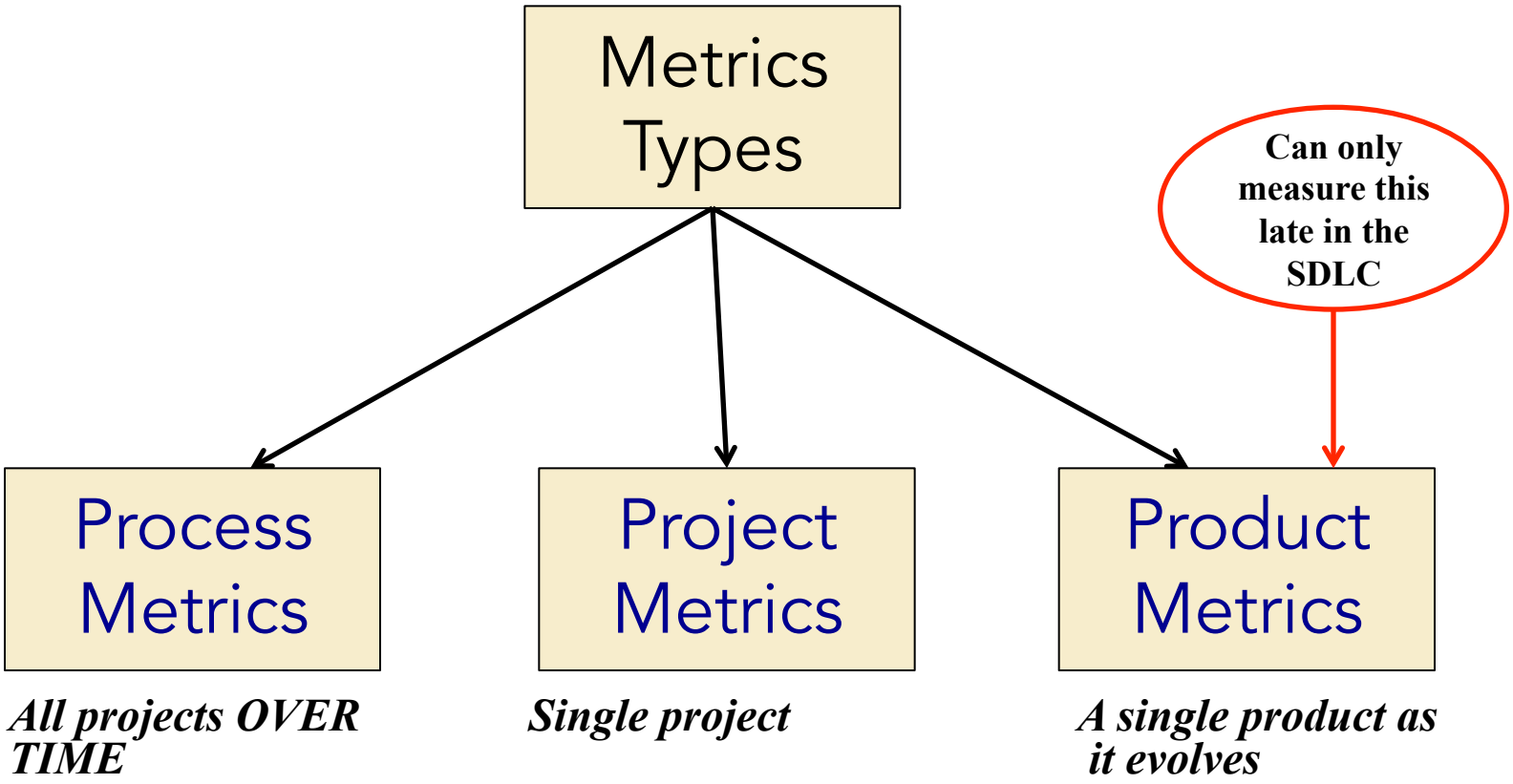
(Lord Kelvin in 1883) and...

"To measure is to know".

Measurement – 2)

- Characterise - understanding of process, products, resources and environment (baselines)
- Evaluate – what is the status of plans
- Predict – better understand relationships
- Improve – bottlenecks/roadblocks, root causes of problems → how to change

Metrics in SE

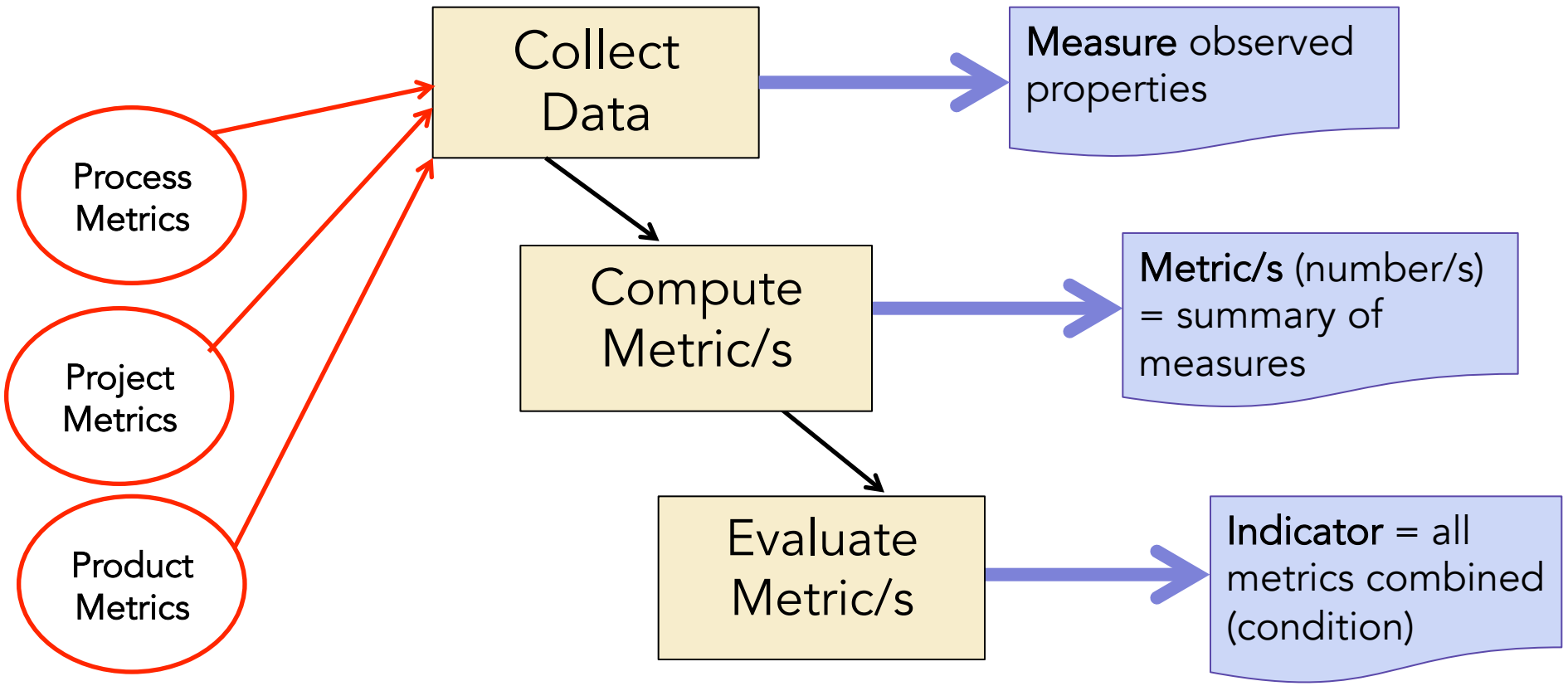


Measuring - indicators

- An indicator is a 'thing' that indicates the state or level of measurement
- PROCESS: efficacy of existing process (i.e. paradigm, SE tasks, work products and milestones)
- PROJECT:

  1) assess the status of an ongoing project
  2) tracking potential risks
  3) uncover problems areas before they go 'critical'
  4) adjust workflow or tasks and
  5) evaluate the team's ability to control quality of the SW work products

The measuring process



TESTING YOU!

- So, when I have collected a single data point, the number of ERRORS uncovered in the review of a single module would be a/an:

1) a MEASURE,
2) a METRIC, or
3) an INDICATOR

When are metrics useful?

1) Simple and computable
2) Empirically and intuitively persuasive
3) Consistent and objective
4) Using consistent units of measurement
5) Programming language independent
6) Useful for feedback

Ex: lines of source code (LOC)

Common metric: LOC (measure lines of code in a product/module)

Get a *measure* of **effort spent to produce** the product and **effort required to understand** the product

Is this entirely accurate? mmm... a good approximation

2 ways to calculate lines of code:

1) *Physical* LOC (including comments and blank lines)
2) *Logical* LOC (ignoring comments and blank lines)

Must be tied to a programming language b/c shape & style of statements differ

Indicator for LOC

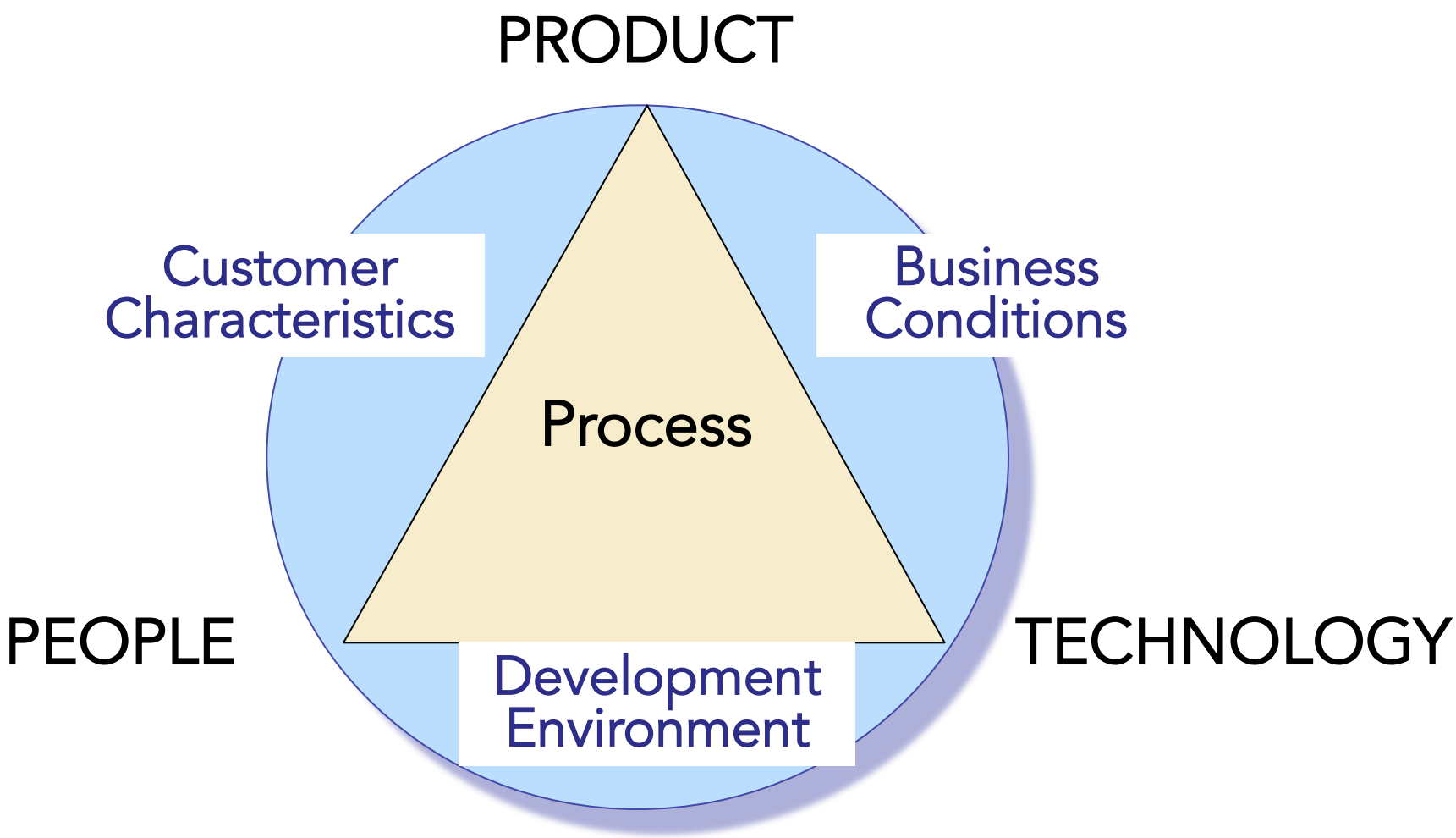
INDICATORS	MEANING
LOC < 1,000	straightforward
1,000 < LOC < 100,000	medium
LOC >= 100,000	difficult

Improving a process?

- How would one go about *improving a specific process?*

1) Identify/develop _____
2) Collect _____
3) Identify _____
4) Collect _____

SW Quality determinants



Process data for quality

- Note that there are 'private' & 'public' uses of process data
- Software Eng's sensitive to metrics therefore metrics should be only visible to the individual SE's: ex's
  - defect rates (by individual)
  - defect rates (by module)
  - errors found during development

Private Process Data philosophy (=PSP)

- *Personal Software Process approach*
- Process descriptions, measurements and methods → to improve SE's personal performance
- Provides: forms, scripts and standards helping SE's to estimate & plan their work
- Shows how to define processes & measure one's qual. & productivity
- Fundamentally: each one is different and method results differ
- Helps SE's to measure & track own work

Use metrics with care

Software Metrics Etiquette:

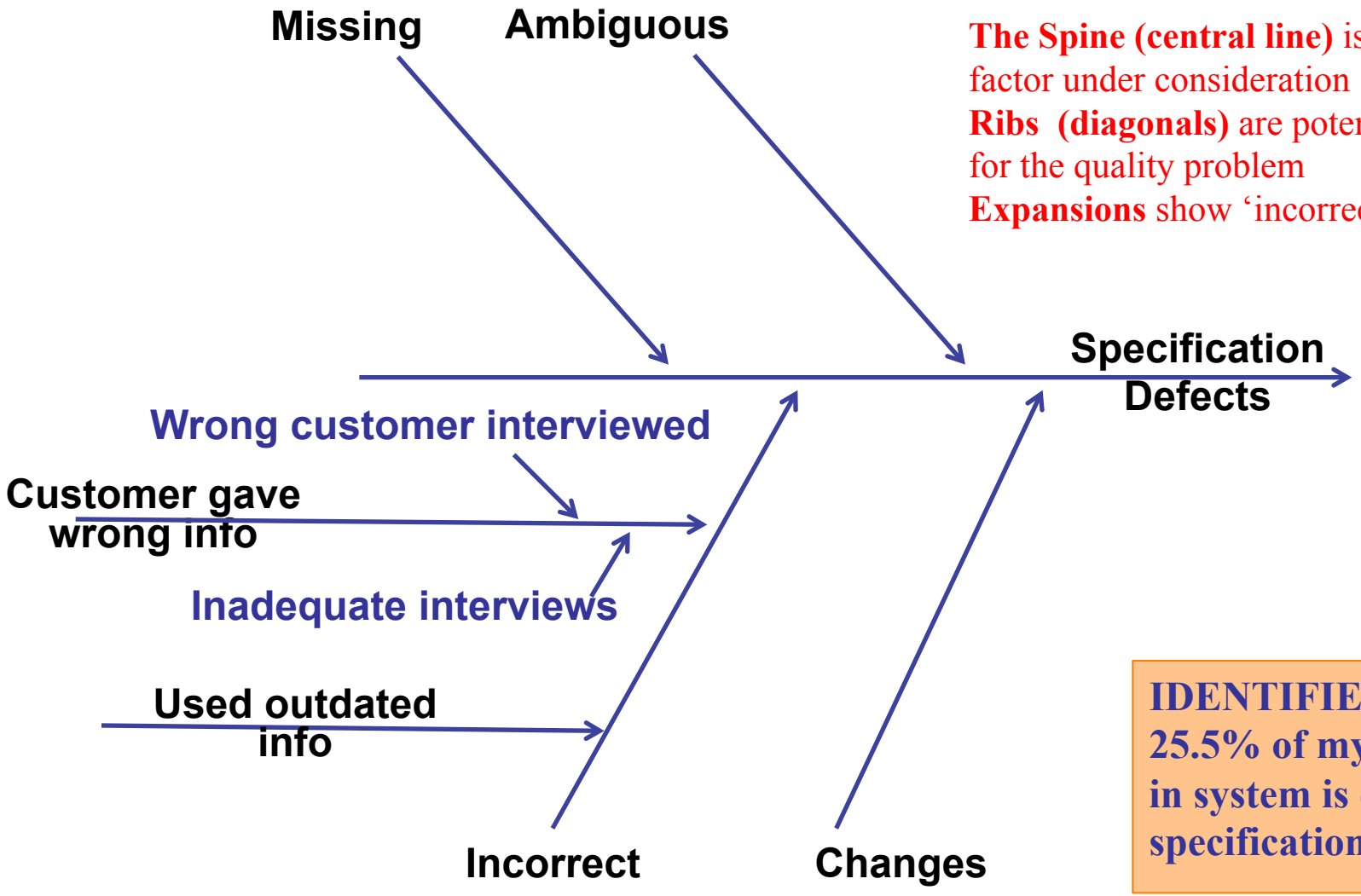
- Common sense and org sensitivity
- Regular feedback to ind's/teams
- Don't use metrics to appraise
- Clear goals & achievable metrics
- Never use to threaten ind's/teams
- Careful with problem area indicators
- Don't obsess single metric for exclusion

Rigorous approach (SSPI)

- *Statistical SW Process Improvement*

1) All errors & defects categorized by origin (flaw in spec, logic, non-conformance to standards)
2) Cost to correct error & defect is recorded
3) No. of errors & defects counted & ranked
4) Overall costs computer per category
5) Results analysed to uncover highest cost
6) Plans developed to modify process in order to eliminate errors & defects

Ex: Defect causes + fishbone diagram



The Spine (central line) is the quality factor under consideration
Ribs (diagonals) are potential causes for the quality problem
Expansions show 'incorrect causes'

IDENTIFIED:
25.5% of my defects in system is due to specification defects

Metrics to Measure Complexity

Three methods...

1) Cyclomatic complexity
2) The CK metrics suite
3) Function point analysis

Measuring complexity

3 Metrics:

1) Cyclomatic complexity
2) The CK metrics suite
3) Function point analysis

1) Cyclomatic complexity

- LOC is one measure (depends on programming language)
- Measure: how many decision points there are in a program
- Depends less on programming lang.

DEF:

Cyclomatic complexity is the upper bound of the number of *linearly independent paths* in the code of a program

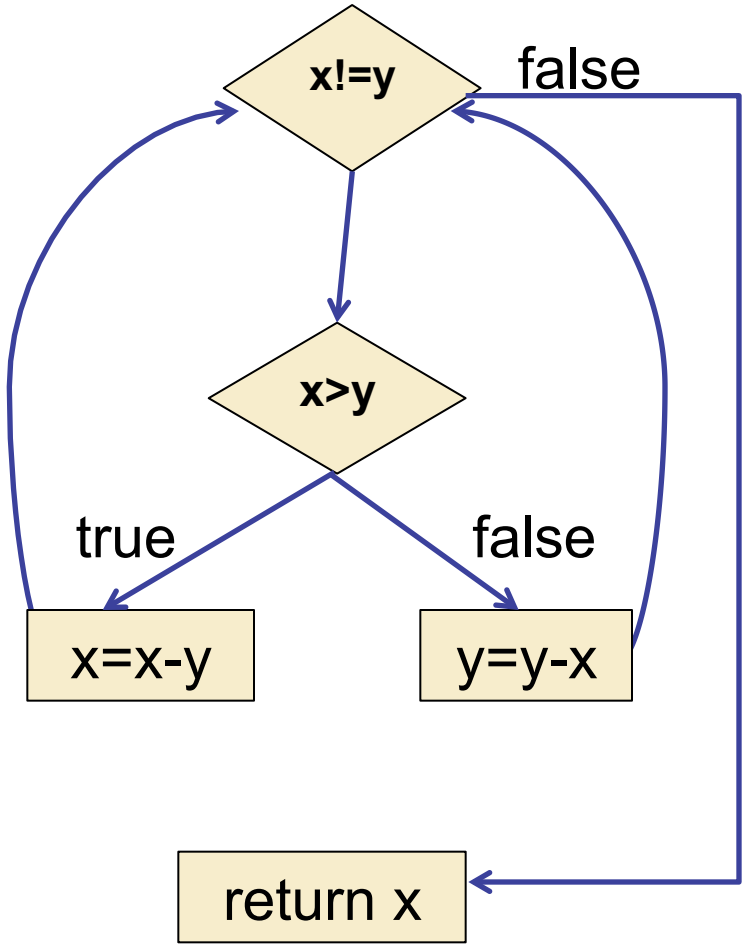
1) Cyclomatic complexity

- Calculating of a *graph*
- First convert the program into its *control flow graph* (a directed graph representing all paths that the program can execute
- Graphs has nodes representing statement and branches, and direct edges representing flow b/w those nodes
- That is: an edge existing b/w 2 nodes is the statements/branches represented by those nodes that can be executed one after the other

Ex Cyclomatic complexity

CODE:

```
int gcd(int x, int y)
{
  while (x != y)  {
    if (x > y)  {
      x – x – y;
    }
    else {
      y = y – x;
    }
  }
  return x;
}
```



McCabe: the upper bound of the number of linearly dependent paths in a program is equivalent to D + 1 where D is the number of decision points

Applications of Cyclomatic complexity

Three typical applications

i) Measuring program complexity: developers to measure code complexity when writing and re-factor code into multiple procedures/ modules when there is high complexity

ii) Testing: give an idea of # test cases to be generated.
If cyclomatic complexity is C, then C is the most # test cases to execute every branch/ decision in a program. And C is least # test techniques that generated a test case for each linearly dependent path in a program

Applications of Cyclomatic complexity

Three typical applications (cont).

iii) Defect estimating: how much a developer and original programmer have to track and examine when attempting to understand a program

Program with High Cyclomatic Complexity is likely more complex to develop

Also contain more faults...
Cyclomatic complexity correlated with fault count in Java classes.

Classes with complexity of 11 had 28% chance of being fault-prone...74 had 98% chance of being fault-prone – so more time on testing....

Questions ?

3 Metrics to measure complexity

1) Cyclomatic complexity ✔
2) The CK metrics suite ✗
3) Function point analysis ✗

Week 8 Lecture 2 . . .

We will continue on Friday with:

2) The CK metrics suite ✗

3) Function point analysis ✗

+

MORE!